Accessing data with MySQL

Gustavo García, 29/03/2023.

Voy a seguir principalmente la guía oficial de Spring:

<https://spring.io/guides/gs/accessing-data-mysql/>

Voy a comentar en diversos lugares, y agregaré otras fuentes.

# Esta es una aplicación web, no una API REST

Por eso, no usamos la anotación @RestController, sino las anotaciones @Controller y @ResponseBody.

Una aplicación web retorna HTM + CSS + JavaScript. Una API REST retorna datos en formato JSON o XML directamente en el response body.

<https://www.javacodegeeks.com/2017/08/difference-restcontroller-controller-annotation-spring-mvc-rest.html>

<https://www.geeksforgeeks.org/difference-between-controller-and-restcontroller-annotation-in-spring/>

# Opciones

Hay varias opciones para acceder a la base de datos. Acá vamos a usar MySQL y JPA.

# Crear el proyecto Spring Boot

Creo el proyecto con la extensión de VS Code para Initializr.

ctrl + shift + p

Java: Create Java project…

Spring Boot

Maven Project

2.7.0

Java

tup

simple

17

Jar

## Dependencias

Spring Web

Spring Data JPA

MySQL Driver

SpringBoot DevTools

# Crear la base de datos y el usuario

El script para crear la base de datos y el usuario está en simple.sql en este mismo directorio. Yo lo usé en phpMyAdmin de WampServer.

# pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-parent</artifactId>

        <version>2.7.0</version>

        <relativePath /> <!-- lookup parent from repository -->

    </parent>

    <groupId>tup</groupId>

    <artifactId>simple</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <name>simple</name>

    <description>Demo project for Spring Boot</description>

    <properties>

        <java.version>17</java.version>

    </properties>

    <dependencies>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-data-jpa</artifactId>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-web</artifactId>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-devtools</artifactId>

            <version>2.7.0</version>

        </dependency>

        <dependency>

            <groupId>mysql</groupId>

            <artifactId>mysql-connector-java</artifactId>

            <scope>runtime</scope>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-test</artifactId>

            <scope>test</scope>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

                <groupId>org.springframework.boot</groupId>

                <artifactId>spring-boot-maven-plugin</artifactId>

            </plugin>

        </plugins>

    </build>

</project>

# src\main\resources\application.properties

# Debe ser igual al artifactId del pom

spring.application.name=simple

server.port=8080

spring.datasource.url=jdbc:mysql://localhost:3306/simple

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username=springuser

spring.datasource.password=ThePassword

spring.jpa.hibernate.ddl-auto=update

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

# src\main\java\tup\simple\SimpleApplication.java

package tup.simple;

/\*

Esta es la clase principal de la aplicación.

Vemos que solo tiene las anotaciones de SpringBoot.

\*/

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SimpleApplication {

    public static void main(String[] args) {

        SpringApplication.run(SimpleApplication.class, args);

    }

}

# src\main\java\tup\simple\controllers\UserController.java

package tup.simple.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import tup.simple.models.User;

import tup.simple.repositories.UserRepository;

@Controller // This means that this class is a Controller

// La URL que vaya en la anotación habrá que agregarla detrás del puerto :8080

// en todas las llamadas a esta aplicación.

// Por ejemplo @RequestMapping("/user") resultaría en lo siguiente:

// localhost:8080/anime.... y detrás de esto habría que agregar el

// resto de la URL.

// En este caso, no necesitamos nada, y queda simplemente localhost:8080

@RequestMapping("")

public class UserController {

    // This means to get the bean called userRepository

    // which is auto-generated by Spring, we will use it to handle the data

    @Autowired

    private UserRepository userRepository;

    @PostMapping("/add") // Map ONLY POST Requests

    public @ResponseBody String addNewUser(@RequestParam String name, @RequestParam String email) {

        // @ResponseBody means the returned String is the response, not a view name

        // @RequestParam means it is a parameter from the GET or POST request

        User n = new User();

        n.setName(name);

        n.setEmail(email);

        userRepository.save(n);

        return "Saved";

    }

    @PostMapping("/delete/{id}") // Map ONLY POST Requests

    public @ResponseBody String deleteUserById(@PathVariable Long id) {

        // @ResponseBody means the returned String is the response, not a view name

        // @RequestParam means it is a parameter from the GET or POST request

        userRepository.deleteById(id);

        return "Deleted";

    }

    @GetMapping("/all")

    public @ResponseBody String getAllUsers() {

        // This returns a JSON or XML with the users

        Iterable<User> iterable = userRepository.findAll();

        String resp = """

                <style>

                #users {

                  font-family: Arial, Helvetica, sans-serif;

                  border-collapse: collapse;

                  width: 100%;

                }

                #users td, #users th {

                  border: 1px solid #ddd;

                  padding: 8px;

                }

                #users tr:nth-child(even){background-color: #f2f2f2;}

                #users tr:hover {background-color: #ddd;}

                #users th {

                  padding-top: 12px;

                  padding-bottom: 12px;

                  text-align: left;

                  background-color: #04AA6D;

                  color: white;

                }

                </style>

                """;

        resp += "<table id ='users'>"

                + "<tr>"

                + "<th>Id</th>"

                + "<th>Name</th>"

                + "<th>Email</th>"

                + "</tr>";

        // No puedo usar forEach() con una función lambda

        // porque el scope de las variables no lo permite.

        for (User user : iterable) {

            resp += "<tr>"

                    + "<td>" + user.getId() + "</td>"

                    + "<td>" + user.getName() + "</td>"

                    + "<td>" + user.getEmail() + "</td>"

                    + "</tr>";

        }

        return resp + "</table>";

    }

    @GetMapping("")

    public @ResponseBody String hola() {

        return "Hola";

    }

}

# src\main\java\tup\simple\models\User.java

package tup.simple.models;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity // This tells Hibernate to make a table out of this class

public class User {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private String name;

    private String email;

    public Long getId() {

        return id;

    }

    public void setId(Long id) {

        this.id = id;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getEmail() {

        return email;

    }

    public void setEmail(String email) {

        this.email = email;

    }

}

# src\main\java\tup\simple\repositories\UserRepository.java

package tup.simple.repositories;

import org.springframework.data.repository.CrudRepository;

import tup.simple.models.User;

// This will be AUTO IMPLEMENTED by Spring into a Bean called userRepository

// CRUD refers Create, Read, Update, Delete

public interface UserRepository extends CrudRepository<User, Long> {

}

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>

# Tests

En este momento, si en la consola sale un mensaje acerca de tests que no se han podido ejecutar, ir a la carpeta:

src\test\java\tup\simple\SimpleApplicationTests.java

y comentar toda la clase. También se puede borrar, pero por las dudas la dejamos. Esto es para que no corra los test, que por ahora no nos interesan.

# Probar

Ya podemos probar con:

<http://localhost:8080>

Veremos la respuesta: Hola.

# Listar usuarios

<http://localhost:8080/all>

# Agregar usuarios

En la consola de Windows, NO en el PowerShell de Visual Studio, ni en el browser:

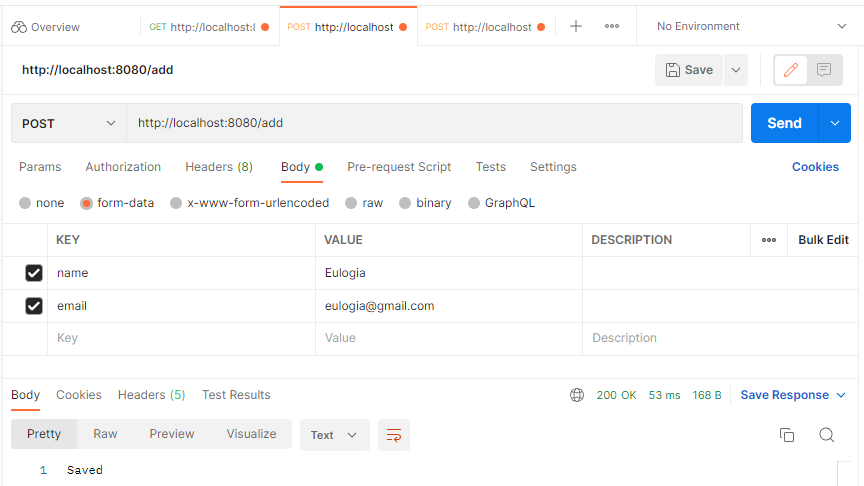
curl localhost:8080/add -d "name=Eulogia" -d "email=eulogia@gmail.com"

curl localhost:8080/add -d "name=Filiberto " -d "email=filiberto@gmail.com"

curl localhost:8080/add -d "name=Lindor" -d "email=lindor@gmail.com"

curl localhost:8080/add -d "name=Nicasio" -d "email=nicasio@gmail.com"

La respuesta es "Saved" en cada caso.

También lo podríamos hacer en PostMan:

Notar que los parámetros se pasan como form data, no como un literal JSON de objeto.

PostMan nos responde Status 200 Ok, y en el panel del body de la response pone Saved, que es lo que retorna el método correspondiente.

# Discusión

La clase principal es SimpleApplication. Lo único que hace, desde nuestro punto de vista, es arrancar la aplicación, de modo que TomCat estará listening en el puerto 8080.

Con la aplicación andando, la clase UserController se hace cargo de los requests. Cuando nosotros escribimos en la consola de Windows:

curl localhost:8080/add -d "name=algún\_nombre" -d "email=algun@gmail.com"

estamos haciendo un request POST a nuestra aplicación.

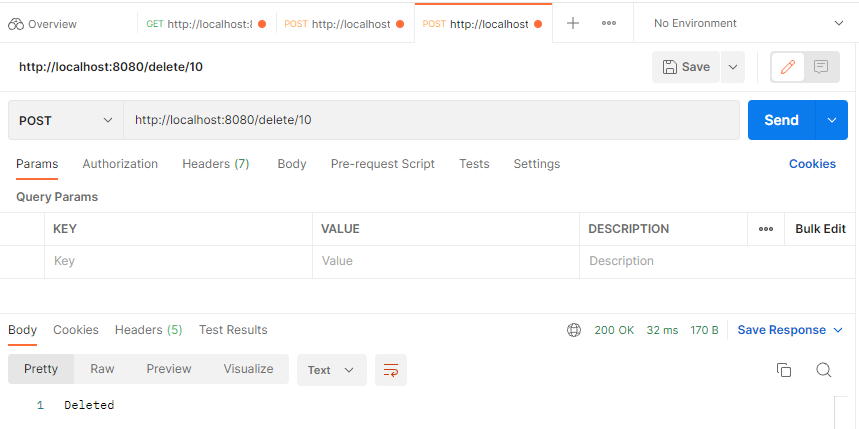
La URL termina con /add. Lo encontramos en la línea 29 de la misma clase:

@PostMapping("/add") // Map ONLY POST Requests

El comentario explica lo que está pasando. La URL que escribimos en la invocación es mapeada al método addNewUser(). Este método toma del body request POST los dos parámetros que tiene como argumentos, crea un nuevo objeto de tipo User, puebla las propiedades usando los parámetros, e invoca al método save() del objeto userRepository, que es de tipo UserRepository. Si vamos a la interfaz UserRepository veremos que es muy corta. Declara que extiende la interfaz CrudRepository, pero no vemos que implemente nada. CrudRepository, a su vez, declara varios métodos pero no los implementa. La anotación @Autowired[[1]](#footnote-1), [[2]](#footnote-2) es la que hace el trabajo. Utiliza un patrón llamado inyección de dependencias, para evitar que escribamos código boiler plate, es decir cosas que son obvias, muy sabidas, repetitivas, que no requieren pensar y se pueden generar automáticamente.

En la documentación de la interfaz CrudRepository[[3]](#footnote-3) está la lista de los métodos disponibles, que se pueden usar sin necesidad de programar la implementación.

## Borrar un usuario por Id

PostMan retorna Status 200 Ok, y el body de la response dice Deleted, que es lo que retorna el método.

1. <https://www.baeldung.com/spring-autowire> [↑](#footnote-ref-1)
2. <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/annotation/Autowired.html> [↑](#footnote-ref-2)
3. <https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html> [↑](#footnote-ref-3)